Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем

Информационные системы и базы данных

Чирков Александр Александрович

Использование методов ML для предсказания сбоев в кластерах хранения данных

Выпускная квалификационная работа

Научный руководитель:
к. ф.-м. н., доцент Графеева Н. Г.

Рецензент:
старший преподаватель Симуни М. Л.

Санкт-Петербург
2017

SAINT-PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems

Information Systems and Databases

Chirkov Aleksandr

Applying ML methods to predict failures in data storage clusters

Graduation Project

Scientific supervisor:

Assoc. Prof. Natalia Grafeeva

Reviewer:

Senior Lecturer Michael Simuni

Saint-Petersburg

2017

Оглавление

Введение

В наше время люди стали осознавать важность накопления данных в различных областях жизни. Данные собираются в различных сферах: бизнес, индустрия, социальная сфера. Но помимо накопления данных, нужно уметь их использовать. Для этого применяются методы машинного обучения, с помощью которых в данных можно находить закономерности и строить выводы.

Для накопления данных используются системы хранения данных (СХД). СХД состоят из большого числа компьютеров, объединенных в кластеры. СХД должны не просто хранить данные, но и обеспечивать их целостность. Данные бывают критически важными, и их потеря может оказать негативное влияние на компанию, в которой эти данные используются. Несмотря на качество оборудования и программного обеспечения в СХД происходят ошибки, которые могут быть вызваны различными причинами и вести к потере данных.

Вся СХД обслуживается различным программным обеспечением, которое записывает информацию о происходящих в системе событиях в текстовые файлы.

Лог – текстовый файл, содержащий строки, описывающие события, произошедшие в системе. Часть сообщений говорит нам об ошибках в системе. Ошибки в системах хранения данных приводят не только к потере данных, но и к другим видам сбоев, например, зависаниям системы. Затем операторы технической поддержки вынуждены вручную перебирать лог в поисках причины произошедшего сбоя. Но к сбоям могут приводить не только конкретные ошибки, а совокупность ошибок или даже совокупность событий, не являющихся ошибками. Поэтому не нужно концентрироваться на поиске конкретных ошибок, нужно искать аномальное поведение системы. И при наличии аномального поведения высылать предупреждение операторам технической поддержки, чтобы они смогли принять меры и предотвратить будущий сбой.

1. Постановка задачи

Целью данной дипломной работы является применение методов машинного обучения для предсказания сбоев в системах хранения данных. Для достижения этой цели в рамках работы были сформулированы следующие задачи.

1) Извлечение шаблонов сообщений в логах, для того, чтобы отделить постоянные значения от переменных.
2) Построение признаков на основе извлеченных шаблонов.
3) Предсказание сбоев на основе построенных признаков.

2. Обзор

2.1 Обзор методов построения признаков и выявления аномального поведения

Выявлять сбои на основе логов можно по-разному.

Самые очевидные способы основаны на построковом анализе лога и нахождении ключевых слов, наподобие error и fault. Но в данном подходе определяются не столько причины сбоев, сколько сами сбои, а главная задача состоит в том, чтобы помочь их предотвратить. Также имеет смысл анализировать группу сообщений в логах, нежели каждое сообщение по отдельности, так как причиной сбоя может быть совокупное влияние нескольких факторов, а появление одной ошибки может не оказать значительного влияния на работу системы.

Для определения сбоев и их причин, можно использовать накопленные данные о сбоях. Авторы [1] использовали данные сообщений в службу поддержки о сбоях. Авторы [2] использовали размеченные файлы логов: файлы, содержащие симптомы сбоев. Для того чтобы предсказать сбой, необходимо уметь строить признаки для группы сообщений, для построения признаков нужны шаблоны, так как сообщения содержат переменные значения. В признаках можно, например, учитывать такие факторы, как частота появления определенного шаблона или его периодичность. Авторы [1] и [2] использовали в дальнейшем классификатор, позволяющий определить аномальное поведение системы, классифицируя группу сообщений на основе построенных признаков. И в том и в другом случае, выявляются лишь сбои, которые уже происходили.

Для построения признаков авторы [3] использовали значения переменных в шаблонах, рассматривая соотношения между ними, так как при нормальном состоянии системы относительная частота каждого значения переменной остается одинаковой и значительно изменяется, когда появляются проблемы. В дальнейшем на основе этого применяется детектор аномалий, чтобы выявить аномальное поведение системы.

2.2 Обзор методов выявления шаблонов в логах

Часто в строках логов можно выделить общую структуру из нескольких полей. В моем случае эта структура выглядит следующим образом:

дата кластер программа сообщение

Если первые 3 поля имеют всегда один и тот же формат, то последнее поле *сообщение*, может быть различного формата. Данное поле сообщает, что же конкретно произошло в системе. Оно содержит, как постоянные, так и переменные значения. Постоянные значения определяют шаблон сообщения.

Выявлять шаблоны можно по-разному:

Авторы [3] для определения шаблонов использовали исходный код программ. Отсюда вытекает очевидный минус – не всегда есть доступ к исходному коду, так как не все используемое программное обеспечение является открытым.

Если доступа к исходному коду программ нет, то остается анализировать собирающиеся логи и на основе этих данных строить шаблоны.

Один из самых первых алгоритмов это SLCT [4]. На основе его построен улучшенный алгоритм LogCluster [5]. Он выделяет самые часто встречающиеся слова и на основе этих слов анализирует каждую строку лога, выделяя шаблоны-кандидаты. Затем отбрасывает нечасто встречающиеся из полученных шаблонов-кандидатов. Из-за того, что алгоритм строит шаблоны на основе часто встречающихся слов, точность шаблонов будет страдать (в часто встречающихся словах может оказаться переменное значение). Алгоритм также имеет очевидный минус: строки, не содержащие частотные слова отбрасываются. А не частотные шаблоны-кандидаты помечаются как выбросы и объединяются в отдельное множество, тем самым исключаются редко встречающиеся шаблоны.

Другой алгоритм IPLoM [6] основан на иерархической кластеризации. Он делит множество сообщений на все меньшие кластеры, пытаясь оставить в

результате в каждом кластере сообщения, которые имеют одинаковый шаблон. Он оставляет редко встречающиеся сообщения, но не учитывает, что переменные значения могут состоять из нескольких слов.
За основу был выбран алгоритм IPLoM и доработан, чтобы решить его существующие проблемы.

3. Исходные данные

Файлы логов системы хранения данных.

Количество сообщений	**Размер данных**
116 336	10.3 МБ

Данные были загружены в таблицу:

id	*raw_msg*
0	*Apr 18 20:04:06* genesis kernel: In lrvm_cleanup_module, line 958: CPU 6 Unregistering RVM client local_rvm
1	*Apr 18 20:04:10* genesis kernel: In lrvm_cleanup_module, line 960: CPU 6 Unregistering block device LocalRVM
…	…

Произведена их предобработка:

1) выделены поля: *дата, кластер, программа, сообщение*
2) добавлено поле *слова*, содержащее список из слов поля *сообщение* (разбиение на слова происходило по пробелам)

id	*date*	*cluster*	*program*	*msg*	*words*
0	*Apr 18 20:04:06*	genesis	kernel	In lrvm_cleanup_module, line 958: CPU 6 Unregistering RVM client local_rvm	[‘In’, ‘lrvm_cleanup_module,’, ‘line’, ‘958:’, ‘CPU’, ‘6’, ‘Unregistering’, ‘RVM’, ‘client’, ‘local_rvm’]

1	*Apr 18 20:04:10*	genesis	kernel:	In lrvm_cleanup_module, line 960: CPU 6 Unregistering block device LocalRVM	[‘In’, ‘lrvm_cleanup_module,’, ‘line’, ‘960:’, ‘CPU’, ‘6’, ‘Unregistering’, ‘block’, ‘device’, ‘LocalRVM’]
…	…	…	…	…	…

4. Извлечение шаблонов из логов

Общая структура каждой строки:

дата кластер программа сообщение

Интерес представляет последнее поле: *сообщение.* Именно в нем содержится ключевая информация.

Шаблон сообщения – это строка, состоящая из обычных слов (постоянные значения) и специальных символов на месте переменных значений.

Пример шаблона сообщения:

In * line * CPU *

4.1 Описание алгоритма IPLoM

4.1.1 Общее описание алгоритма:

IPLoM состоит из 3 шагового процесса иерархической кластеризации.

На каждом шаге алгоритма сообщения делятся на все меньшие кластеры.

В результате последнего шага, каждый кластер должен содержать сообщения, максимально хорошо удовлетворяющие одному шаблону.

4 шаг алгоритма – формирование шаблона для каждого кластера.

4.1.2 Основные шаги алгоритма:

1) На входе шага 1: таблица со всеми сообщениями лога.

   В шаге 1 происходит кластеризация на основе количества слов в сообщении.

   Для каждой строки лога для поля *сообщение* считаем количество слов и объединяем строки с одинаковым количеством, формируя кластеры.

   Удаляем кластеры с количеством слов равным 0 и 1.

```
# df – таблица с предобработанными данными
# для каждой строки таблицы считаем количество слов в поле words и присваиваем это значение полю event_size
df['event_size'] = df['words'].map(lambda x: len(x))
# группируем строки на основе значения поля event_size
step1gr = df.groupby(['event_size'])
# создаем из групп новую таблицу с иерархическим индексом, состоящим из полей event_size, id
for name,group in step1gr:
        group = group.set_index(['event_size',group.index])
        step1df = concat([step1df,group])
# удаление кластеров с количеством слов равным 0 и 1
step1df.drop(0,level=0,inplace=True)
step1df.drop(1,level=0,inplace=True)
# в итоге step1df – таблица с иерархическим индексом, содержащая получившиеся кластеры
```

Итог – таблица следующего вида:

event_size	*id*	*words*
…	…	…
3	15	[‘writing’, ‘file’, ‘successful’]
	46	[‘writing’, ‘file’, ‘aborted’]
	50	[‘writing’, ‘block’, ‘successful’]
	53	[‘reading’, ‘cache’, ‘successful’]
4	7	[‘Installing’, ‘public’. ‘key’, ‘data’]
…	…	…

2) На входе шага 2: таблица с кластерами из шага 1.

   Рассматриваем каждый кластер по отдельности.

Шаг 2 основывается на предположении, что позиция в сообщениях кластера с минимальным количеством различных значений обычно должна содержать постоянное значение.

Поэтому находим позицию сообщений кластера с минимальным количеством различных значений и кластеризуем на основе этой позиции (для каждого значения – свой кластер).

```
# рассматриваем каждый кластер по отдельности
for group in step1df.groupby(level=0):
   # подсчитываем количество различных значений в     каждой позиции
   val_count = val_list(group)
   # находим индекс минимального значения
   min_ind = val_count.index(min(val_count))
   # для каждого сообщения кластера в поле min_val_word записываем слово
из найденной позиции
   group['min_val_word'] = group['words'].map(lambda x: x[min_ind])
   # создаем иерархический индекс из полей event_size, min_val_word, id
   group = group.set_index(['event_size','min_val_word','id'])
   # добавляем к новой таблице группу с новым индексом
   step2df = concat([step2df,group])
```

Рассмотрим кластер с event_size равным 3:

event_size	*id*	*words*
…	…	…
3	15	[‘writing’, ‘file’, ‘successful’]
	46	[‘writing’, ‘file’, ‘aborted’]
	50	[‘writing’, ‘block’, ‘successful’]
	53	[‘reading’, ‘cache’, ‘error’]
…	…	…

Для каждой позиции подсчитаем количество различных значений:

Для 1 позиции – 2

Для 2 позиции – 3

Для 3 позиции – 3

1 позиция будет выбрана, как позиция с минимальным количеством различных значений и для каждого из значений будет создан отдельный кластер.

event_size	*min_val_word*	*id*	*words*
…	…	…	…
3	writing	15	[‘writing’, ‘file’, ‘successful’]
		46	[‘writing’, ‘file’, ‘aborted’]
		50	[‘writing’, ‘block’, ‘successful’]
	reading	53	[‘reading’, ‘cache’, ‘error’]
…	…	…	…

3) На входе шага 3: таблица с кластерами из шага 2.

   На шаге 3 отбираются 2 позиции и рассматриваются отношения между словами из множеств значений на этих 2 позициях. На основе найденных отношений выделяются новые кластеры.

   Перед началом основного алгоритма шага 3 происходит проверка на случай, если после шага 2 выделились достаточно хорошие кластеры. Для этого выполняются следующие шаги:

   1) Нахождение числа различных значений для каждой позиции сообщений кластера.
   2) Если отношение числа позиций, в которых стоит 1, к количеству слов в сообщении превосходит заданный порог, то кластер считается хорошим и не обрабатывается алгоритмом шага 3.

      Вычисляется соотношение cgr (Cluster Goodness Ratio):

$$cgr = \frac{count(PositionsWith1)}{count(Positions)}$$

Задается константа ct (Cluster Threshold).

Если cgr≥ct, то кластер хороший.

```
function good_cluster_action(group):
   # создаем поле со значением ‘good_cluster’
   group['bij'] = 'good_cluster'
   group = group.reset_index()
   # заново индексируем кластер создавая новый уровень иерархического индекса,
и добавляем в новую таблицу
   group = group.set_index(['event_size','min_val_word','bij','id'])
   step3df = concat([step3df,group])
```

Основной алгоритм шага 3:

1) Нахождение числа различных значений для каждой позиции сообщений кластера.
2) Нахождение наиболее часто встречающегося числа значений. Число должно быть больше чем 1. Обозначим его за maxfreq_token_count. Это число показывает количество различных шаблонов, которые находятся в кластере. Выбираем первые 2 позиции с этим числом.
3) Определение отношения между элементами множеств, находящихся на 2 позициях, выбранных на предыдущем шаге. Для этого изначально создаем словарь для каждой из 2 позиций, где ключ – это слово в одной из позиций, а значение – множество слов, стоящих с ним в паре во второй позиции. В псевдокоде словари – p1dict и p2dict.

   Виды отношений:

   1) Биекция (1-1)

      Элементу первого множества соответствует один элемент второго множества.

Например, если кластер содержит строки:

	Позиция 1	Позиция 2	Позиция 3
	writing	file	successful
	writing	file	aborted
	reading	cache	error
Количество различных значений	2	2	3

Выбранными позициями, будут позиции 1 и 2.

Слова writing и file находятся в отношении биекции.

И слова reading и cache находятся в отношении биекции.

```
function relation_oneone(p1dict,p2dict):
   # создаем множество, в котором будем хранить пары биективных слов
   r11 = set()
   for word_key,word_set in p1dict.items():
      # если мощность множества равна 1, то рассматриваем этот элемент
множества в качестве ключа во втором словаре, и выясняем
состоит ли множество по этому ключу из 1 элемента
      if len(word_set)==1:
         word_key2 = list(word_set)[0]
         if len(p2dict[word_key2])==1:
            r11.add((word_key,word_key2))
   # удаляем найденные отношения из словарей
   for (k1,k2) in r11:
      del p1dict[k1]
      del p2dict[k2]
   return r11
```

2) 1-M или M-1

1-M: элементу первого множества соответствует несколько элементов второго множества.

M-1: наоборот

Например, если кластер содержит строки:

	Позиция 1	Позиция 2	Позиция 3
	writing	file	successful
	writing	cache	aborted
	reading	block	successful
	sending	block	aborted
Количество различных значений	3	3	2

Выбранными позициями будут позиции 1 и 2.

Слову writing соответствуют слова file и cache и это отношение 1-M.

Слову reading соответствует слово block, и слову sending соответствует слово block – это отношение M-1.

```
function relation_onem(p1dict,p2dict):
   # словарь для отношения M-1
   rM1 = {}
   for word_key,word_set in p1dict.items():
      # выясняем состоит ли множество по ключу word_key из 1 элемента,
если да, то рассматриваем этот элемент в качестве ключа во втором
словаре (word_key2), и если множество по этому ключу во втором словаре
состоит не из 1 элемента, то рассматриваем каждый элемент этого
множества, как ключ в первом словаре, и смотрим состоит ли множество по
этому ключу из 1 элемента и является ли этот элемент ключом word_key2
      if len(word_set)==1:
         word_key2 = list(word_set)[0]
         word_set2 = p2dict[word_key2]
         if len(word_set2)!=1:
            for v in word_set2:
               if not (len(p1dict[v])==1 and list(p1dict[v])[0]==word_key2):
                  break
               else:
                  rM1[word_key2] = p2dict[word_key2]
   # удаляем найденные отношения из словарей
   for a,b in rM1.items():
      for v in b:
         del p1dict[v]
      del p2dict[a]
   return rM1
```

3) M-M:

Перекрестные отношения элементов

Например, если кластер содержит строки:

	Позиция 1	Позиция 2	Позиция 3
	writing	file	successful
	writing	cache	aborted
	reading	file	error
Количество различных значений	2	2	3

Выбранными позициями будут позиции 1 и 2.

Слово writing находится в отношении со словами file и cache, при этом слово file также находится в отношении со словом reading.

```
function relation_mm(p1dict,p2dict):
    # в отношение M-M попадают слова, неотобранные для отношений 1-1, 1-M и M-1
   s1 = set(p1dict.keys())
   s2 = set(p2dict.keys())
   return (s1,s2)
```

4) Кластеризация на основе вида отношения
   1) Биекция

      Для каждой биекции строим отдельный кластер.

```
function oneone_action(r11,group):
   for (k1,k2) in r11:
      # поля w1 и w2 – поля таблицы, которые для каждой строки содержат слова, находящиеся на первой и второй выбранной позиции соответственно
      # выбираем строки, удовлетворяющие биективному отношению
      group_filt = group[(group['w1']==k1) & (group['w2']==k2)]
      # новый ключ состоит из конкатенации слов отношения
      group_filt['bij'] = k1+k2
      group_filt = group_filt.reset_index()
      # заново индексируем группу выбранных строк, формируя из них новый кластер и добавляя в новую таблицу
      group_filt = group_filt.set_index(['event_size','min_val_word','bij','id'])
      step3df = concat([step3df,group_filt])
```

   2) 1-M или M-1

      В зависимости от того является ли M множеством постоянных значений или множеством переменных значений:

      1) Постоянные значения:

         Для каждого значения отдельный кластер

      2) Переменные значения:

         Для всех один кластер

      Для проверки на вид множества M, вычисляется следующее отношение (Rank Position):

$$rp = \frac{count(M)}{count(MessagesWithWordsInM)}$$

То есть отношение числа значений в M, к размеру группы сообщений со словами из M в выбранной позиции.

Задаются две константы ub (UpperBound) и lb (LowerBound). Если rp>ub, то в M переменные значения, если же rp<lb, то в M постоянные значения.

```
function mone_action(rM1,group,ub,lb):
   for k,v in rM1.items():
      # вычисляем rank position
      rp = len(v)/len(group[group['w1'].isin(v)])
      # переменные значения, значит сообщения в 1 кластер
      if rp>ub:
         group_filt = group[(group['w1'].isin(v)) & (group['w2']==k)]
         group_filt['bij'] = k+list(v)[0]
         group_filt = group_filt.reset_index()
         group_filt = group_filt.set_index(['event_size','min_val_word','bij','id'])
         step3df = concat([step3df,group_filt])
      # постоянные значения, значит для каждого значения новый кластер
      elif rp<lb:
         for val in v:
            group_filt = group[(group['w1']==val) & (group['w2']==k)]
            group_filt['bij'] = k+val
            group_filt = group_filt.reset_index()
            group_filt =
group_filt.set_index(['event_size','min_val_word','bij','id'])
            step3df = concat([step3df,group_filt])
```

3) M-M:

Все в один кластер.

```
function mm_action(mm_w1,mm_w2,group):
   group_filt = group[(group['w1'].isin(mm_w1)) & (group['w2'].isin(mm_w2))]
   group_filt['bij'] = 'mmrel'
   group_filt = group_filt.reset_index()
   group_filt = group_filt.set_index(['event_size','min_val_word','bij','id'])
   step3df = concat([step3df,group_filt])
```

5) Формируем шаблон для каждого кластера:

   Если количество различных значений в позиции равно 1, то это постоянное значение, иначе переменное.

   При формировании шаблона на месте переменного значения ставим *, при этом храним набор значений переменной.

Код для 3 шага:

```
for group in step2df.groupby(level=[0,1]):
    k = event_size of group
    # val – список множеств значений в каждой из позиций
    # val_count – список числа значений в каждой из позиций
    val, val_count = val_list(k,group)
    # считаем Cluster Goodness Ratio
    cgr = val_count.count(1)/k
    if cgr>=ct:
        # добавляем кластер в новую таблицу с индексным значением
'good_cluster'
        good_cluster_action(group)
        continue
    # находим 2 позиции для поиска отношений
    p1,p2 = find_pos(val_count)
    # создаем словари, где ключ – это слово в одной из позиций, а значение
– множество слов, стоящих с ним в паре во второй позиции
    p1dict,p2dict = find_dict(group,p1,p2)
    # обрабатываем биекции
    r11 = relation_oneone(p1dict,p2dict)
    oneone_action(r11,group)
    # обрабатывем отношения 1-M и M-1
    rM1 = relation_onem(p1dict,p2dict)
    r1M = relation_onem(p2dict,p1dict)
    mone_action(rM1,group,ub,lb)
    onem_action(r1M,group,ub,lb)
    # обрабатывем отношения M-M
    mm_w1,mm_w2 = relation_mm(p1dict,p2dict)
    mm_action(mm_w1,mm_w2,group)
```

4.2 Доработка алгоритма

Одним из слабых мест алгоритма является нахождение позиций для поиска отношений в шаге 3, так как не всегда можно найти 2 позиции, которые удовлетворяют условию, что в них будет максимально часто встречающееся число значений, больше 1.

Для улучшения поиска необходимо рассмотреть несколько случаев:

1) Если максимальная частота равна 1, значит каждое число значений встречается всего 1 раз. Тогда необходимо отсортировать словарь с числом значений по возрастанию. Если первый по возрастанию элемент равен 1, и элементов в словаре больше 2, тогда взять элементы на второй и третьей позициях, иначе на двух первых.
2) Если максимальная частота больше 1, то возникает также 2 случая. Если число с максимальной частотой это 1 и элементов в словаре больше 2, то убрать 1 из рассмотрения и взять две первых позиции из отсортированного списка. Иначе же взять 2 позиции с найденным числом значений с максимальной частотой.

```
function find_pos(val_count):
    # val_count – список с числом значений в каждой из позиций
    # считаем частоту каждого числа значений
    freq_occ_tokens = {}
    for count_token in val_count:
        if count_token in freq_occ_tokens:
            freq_occ_tokens[count_token] += 1
        else:
            freq_occ_tokens[count_token] = 1
    # максимальная частота
    max_freq_occ = 0
    # максимально часто встречающееся число значений
    maxfreq_token_count = 0
    # ищем максимально часто встречающееся число значений
    for count_token,freq_occ in freq_occ_tokens.items():
        if freq_occ>max_freq_occ:
            max_freq_occ = freq_occ
            maxfreq_token_count = count_token
    # случай, когда максимальная частота равна 1
    if max_freq_occ == 1:
        sort_keys = sorted(freq_occ_tokens)
        if sort_keys[0] == 1 and len(freq_occ_tokens)>2:
            p1count = sort_keys[1]
            p2count = sort_keys[2]
        else:
            p1count = sort_keys[0]
            p2count = sort_keys[1]
        p1 = val_count.index(p1count)
        p2 = val_count.index(p2count)
    else:
        # случай, когда максимально часто встречающееся число значений равно 1 и
различных чисел больше 2
        if maxfreq_token_count == 1 and len(freq_occ_tokens)>2:
            del freq_occ_tokens[1]
            sort_keys = sorted(freq_occ_tokens)
            p1count = sort_keys[0]
            p1 = val_count.index(p1count)
            p2count = sort_keys[1]
            p2 = val_count.index(p2count)
        else:
            p1 = val_count.index(maxfreq_token_count)
            temp_c = val_count[p1+1:]
            p2 = temp_c.index(maxfreq_token_count) + p1 + 1
    return (p1,p2)
```

4.3 Улучшение алгоритма

Слабое место алгоритма, это то, что он не учитывает, что переменные значения могут состоять из нескольких слов, для этого после нахождения шаблонов, можно воспользоваться идеей нахождения частотных слов и так как у нас уже готовые шаблоны, то частотные слова хорошо отделят переменные значения от постоянных, и тем самым мы сможем объединить несколько готовых шаблонов, для повышения точности алгоритма. Это предельно важно, так как необходимо получить все возможные значения переменной для будущего построения признаков.

Пусть, например, среди шаблонов есть следующие:

status * of sending

status not * of sending

Определить частотные слова of, sending и status и объединить эти 2 шаблона в один:

status * of sending

Описание алгоритма:

1) Задаем пороговое значение для определения частотных слов.
2) Находим частотные слова только среди постоянных значений.
3) Для каждого шаблона создаем ключ, состоящий из частотных слов.
4) Объединяем те шаблоны, ключи которых совпадают. При этом объединяем множества переменных значений.

5. Построение признаков

Построение признаков основывается на том, что по переменной состояния (переменная с небольшим количеством различных значений), можно определить аномальное поведение системы, так как соотношение между различными значениями переменной состояния остается стабильным во время нормального состояния системы и изменяется значительно, когда появляются проблемы.

Чтобы определить является ли переменная переменной состояния, считается отношение числа различных значений переменной к числу строк с этим шаблоном.

В своих экспериментах пороговое значение для этого числа я установил в 0.005.

Если вычисленное отношение меньше или равно порогового значения, то это переменная состояния.

Для построения признаков необходимо найти часто встречающиеся шаблоны, содержащие переменные состояния.

В своих экспериментах я искал шаблоны, которые составляют хотя бы 15% всех сообщений лога. Затем необходимо разделить все сообщения на временные окна, так чтобы шаблон был хотя бы в 80% всех окон.

Построение признаков для 1 такого шаблона:

Для каждого временного окна строится вектор, где каждая компонента вектора – это количество появлений определенного значения переменной в данном временном окне.

На выходе получаем матрицу размера $m \times n$, где m – количество временных окон, n – количество различных значений.

6. Определение аномального поведения.

Для определения аномального поведения используется метод главных компонент.

Его суть заключается в построении гиперплоскости, такой что сумма расстояний от всех объектов до гиперплоскости будет минимальной.

Можно рассмотреть пример в двумерном пространстве.

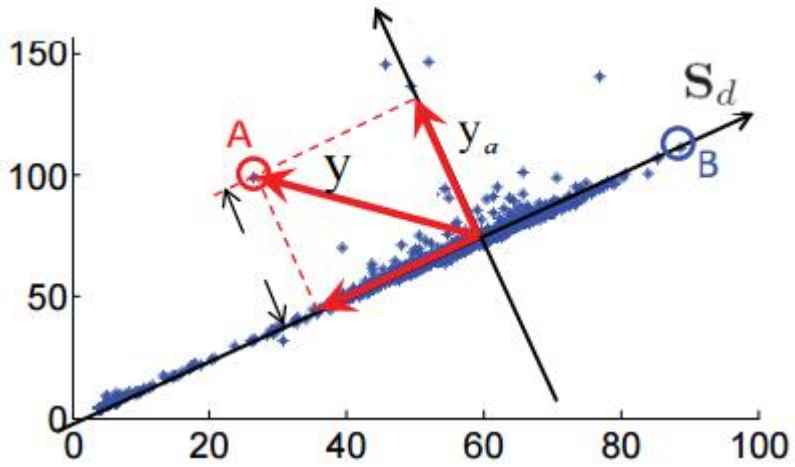
Здесь необходимо провести прямую, так чтобы сумма расстояний от точек до прямой была минимальной.

S_d – подпространство (прямая), задающая нормальное состояние системы.

Чем дальше точка от своей проекции на прямой, тем более аномальной она является.

Для определения аномальности точки задается порог Q (среднее расстояние от всех точек до прямой).

Если расстояние от точки до прямой превышает данный порог, то точка является аномальной.



7. Эксперименты

7.1 Определение шаблонов

Для экспериментов были отобраны 2000 строк и для них вручную выделены шаблоны.

Параметры алгоритма:

Cluster Threshold = 0.9

Upper Bound = 0.9

Lower Bound = 0.1

Для оценки вычислялись следующие величины:

TP – количество шаблонов, которые полностью совпали

FP – количество лишних шаблонов, которые вернул алгоритм

FN – количество шаблонов, которые алгоритм определить не смог

Для оценки алгоритмов вычисляются Precision и Recall:

$Precision = \frac{TP}{TP+FP}$ – точность

$Recall = \frac{TP}{TP+FN}$ – полнота

	Precision	Recall
IPLoM	0,85	0,89
IPLoM (улучшенная версия)	0,94	0,89

Таким образом точность улучшенной версии алгоритма превосходит точность обычной версии.

7.2 Построение признаков и определение аномального поведения.

Для построения признаков алгоритм отобрал один шаблон, в котором переменная состояния имела 98 различных значений.

Длина одного временного окна – 1000 сообщений.

Для оценки вычислялись следующие величины:

TP – временное окно определено аномальным и действительно является аномальным

FP – временное окно определено аномальным, но аномальным не является

FN – аномальные временные окна, которые не определил алгоритм

Для оценки алгоритмов вычисляются Precision и Recall:

$Precision = \frac{TP}{TP+FP}$ – точность

$Recall = \frac{TP}{TP+FN}$ – полнота

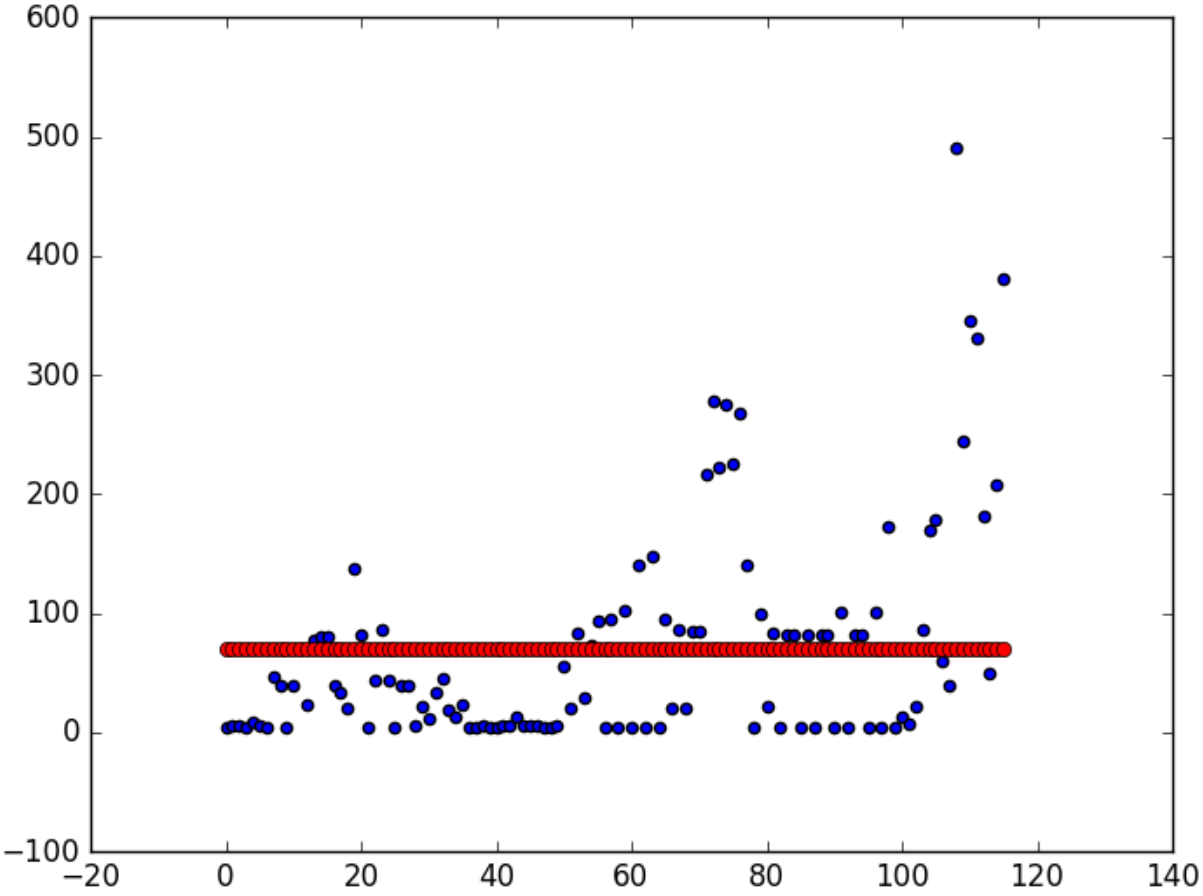
Precision	Recall
0,91	0,88

График расстояний:

По оси X – порядковый номер временного окна

По оси Y – расстояние от точки до ее проекции на гиперплоскость

Красная линия – пороговое значение.



Заключение

В ходе данной работы получены следующие результаты.

- Реализован алгоритм IPLoM для определения шаблонов в логах
- Алгоритм IPLoM доработан и улучшен, произведено сравнение со старой версией с помощью метрик точности и полноты
- Для логов построены признаки на основе найденных шаблонов
- Применен метод главных компонент для определения аномального поведения
- Реализация всех алгоритмов произведена на языке python

Список литературы

[1] Tatsuaki Kimura, Akio Watanabe, Tsuyoshi Toyono, Keisuke Ishibashi, "Proactive failure detection learning generation patterns of large-scale network logs," 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7367332/.

[2] Thomas Reidemeister, Mohammad A. Munawar, Paul A.S. Ward, "Identifying symptoms of recurrent faults in log files of distributed information systems," 2010. [Online]. Available: http://ieeexplore.ieee.org/document/5488459/.

[3] Wei Xu, Ling Huang, Armando Fox, David Patterson, Michael I. Jordan, "Detecting large-scale system problems by mining console logs," 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1629587.

[4] R. Vaarandi, "Mining Event Logs with SLCT and LogHound," 2008. [Online]. Available: https://ristov.github.io/publications/slct-loghound-noms08-web.pdf.

[5] Risto Vaarandi, Mauno Pihelgas, "LogCluster - A data clustering and pattern mining algorithm for event logs," 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7367331/.

[6] A. N. Z.-H. E. E. M. Adetokunbo Makanju, "A Lightweight Algorithm for Message Type Extraction in System Application Logs," 2011. [Online]. Available: http://ieeexplore.ieee.org/document/5936060/.

[7] "Matplotlib," [Online]. Available: https://matplotlib.org/api/mlab_api.html.

[8] "Метод главных компонент," [Online]. Available: https://www.coursera.org/learn/unsupervised-learning/lecture/Famz8/mietod-ghlavnykh-komponient-postanovka-zadachi.

[9] "Sensitivity and specificity," [Online]. Available: https://en.wikipedia.org/wiki/Sensitivity_and_specificity.